Prolog Interpreter in Rewriting Logic

Musab Al-Turki and Ralf Sasse

Based upon: The Art of Prolog by Leon Sterling and Ehud Shapiro

Introductory Prolog Class CS422

Outline

Motivation

- Logic as a formal way of thinking: Goals, Knowledge, Assumptions.
- Deduce consequences from premises.
- Truth/Falsity of statements.
- Logic Programming provides means to execute these ideas.
- Standard Programming Languages (PLs) are close to von-Neumann architecture, Logic Programming abstracts from that.
- Programming as a intellectual exercise, not mindless coding.

Example: Facts and Queries

Database:

```
father(terach,abraham).
father(terach,nachor).
father(terach,haran).
father(abraham,isaac).
female(sarah).
female(milcah).
female(yiscah).
```

Queries:

```
father(terach, nachor)?
father(terach, X)?
father(haran, X)?
male(abraham)?
```

Substitutions and Instances

Substitution:

- Finite set of pairs, mapping variables to terms.
- The mapping is a function!
- No recursion within the set, i.e. variables on the left-hand side only appear on the left-hand side.

Example:
$X = a, Y = Z$ is a valid substitution.
$X = Y, Y = Z$ is not a valid substitution.
Instances:

- A is an instance of B if there is a substitution θ such that $A = \theta(B)$.

Queries and Rules

Existential Queries:

```
father(terach, X)?
```

Conjunctive Queries:

```
father(terach, X) , father(X, lot)?
```

Rules:

```
A <- B, C, D,...
```

Facts are a special case of rules, with empty right-hand side.

```
son(X, Y) <- father(Y, X), male(X).
```

Rule application

Modus ponens applies!

```
A <- B, C, D
B
C
D
```

We can then deduce `A`.
Actually this works on instantiations of all these with proper substitutions.

Meaning of a Logic Program

The *meaning* of a program `P` is the set of ground goals deducible from `P`.

If it is not in your database it is implicitly false!

Prolog

- Sequential implementation of Logic Programming.
- Determinize choices in ground instantiations.
- Determinize which goal to prove next.

Implemented by:

- Unification.
- Depth-first search.

Notation of Rules:

```
A :- B, C, D,... .
```

Prolog Examples

```
[
nat(z) .
nat(s(X)) :- nat(X) .

plus(z, X, X) :- nat(X) .
plus(s(X), Y, s(Z)) :- plus(X,Y,Z)
]

; plus(z, s(z), s(z)) ? .
```

Prolog Examples

```
[
nat(z) .
nat(s(X)) :- nat(X) .

plus(z, X, X) :- nat(X) .
plus(s(X), Y, s(Z)) :- plus(X,Y,Z) .

times(z, X, z) :- nat(X) .
times(s(X), Y, Z) :-  times(X, Y, XY) , plus(XY, Y, Z)
]

; times(s(s(z)), s(s(z)), s(s(s(s(z))))) ?
```

Prolog Examples

```
[father(terach,abraham) . father(terach,nachor) .
father(terach,haran) .   father(abraham,isaac) .
father(haran,lot) .      father(haran,milcah) .
father(haran,yiscah) .   mother(sarah,isaac) .
male(terach) .           male(abraham) .
male(nachor) .           male(haran) .
male(isaac) .            male(lot) .
female(sarah) .          female(milcah) .
female(yiscah) .
parent(P,X) :- father(P,X) .
parent(P,X) :- mother(P,X) .
daughter(D,P) :- parent(P,D) , female(D) .
grandparent(G,X) :- parent(G,P) , parent(P,X) .
] ;  grandparent(terach, X) ?
```

Unification

- For two terms t and t' a substitution θ is called a *unifier* if $\theta(t) = \theta(t')$.
- A unifier θ is more general than a unifier σ if there exists a substitution δ s.t. $\theta; \delta = \sigma$.
- A most general unifier (*mgu*) is a unifier that is more general than any other unifier.

Unification problem similar to the general unification problem (e.g. equational unification):

- Given 2 terms t and t' compute the mgu of t and t'.

Unification Algorithm

Keep a control structure, no details here.
List of constraints $t = t'$ to be solved.
Case analysis:

- $x = t$: if x does not occur in t, substitute t for x in the constraint list.
- $t = t$: just drop this.
- $f(t_1, ..., t_n) = f(s_1, ..., s_n)$ with $n > 0$ then add $t_i = s_i \forall i$ to the constraint list.
- otherwise: failure of unification!

Renaming

Example database:

```
g(Y).
f(X).
```

Query: `f(g(X))?` fails.
Query: `f(g(Z))?` works.

Renaming needs to be applied at every unification step in general. One initial renaming is not enough (recursion!).

Search Motivating Example

Database:

```
1: y(5).
2: y(7).
3: z(7,0).
4: x(M,N) :- y(M), z(M,N).
```

Query: `x(A,B)`

Difficulties in Practice

- Need a stack of the current goal list.
- That stack needs to store (for each of its elements) a stack of rules that have not yet been tried. These need to be in the order given in the database!

Our solution:

- Number the rules; use the number of the rule to be checked next instead of a whole stack of unchecked rules.

Each element of our stack now has this form:

```
{Fl:FactList, N:Nat, S:SubstitutionList}
```

where `Fl` are the open goals, `N` is the pointer into the database and `S` is the substitution gathered until here.

Cut Rule

Cut is used to restrict choices:

```
a(0).
a(1).
b(0).
b(1).
f(X) :- a(X) , ! , b(X) .
```

The cut rule is quite simple to implement:

```
eq {(!, Fl:FactList), 1, Su:Subst} | St:Stack
 = {Fl:FactList, 1, Su:Subst} | noStack .
```

Conclusions

- Implementing Prolog in rewriting logic was quite easy.
- We did not follow the continuation style!
- Two main parts:
  - Unification with substitution, constraint solving, renaming, etc.
  - Search with backtracking, depth-first.

Limitations:

- For every query the whole database needs to be supplied.
- Pre-processing time necessary every run.

Future Work:

- Negation.
- Fancy interface with proper input/output in Prolog style, using the Maude bubble.